**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по домашнему заданию

«Пример многопоточного поиска в текстовом файле с использованием технологии Windows Forms»

|  |  |  |
| --- | --- | --- |
| Выполнила: |  | Проверил: |
| студентка группы ИУ5-33 |  | преподаватель каф. ИУ5 |
| Беспалова У.А. |  | Гапанюк Ю.Е. |
| Подпись и дата:  25.11.2018 |  | Подпись и дата: |

Москва, 2018 г.

Содержание задания

Разработать программу, реализующую многопоточный поиск в файле.

1. Программа должна быть разработана в виде приложения Windows Forms на языке C#. По желанию вместо Windows Forms возможно использование WPF.
2. В качестве основы используется макет, разработанный в лабораторных работах №4 и №5.
3. Реализуйте функцию поиска с использованием расстояния Левенштейна в многопоточном варианте. Количество потоков для запуска функции поиска вводится на форме в поле ввода (TextBox).
4. Реализуйте функцию записи результатов поиска в файл отчета. Файл отчета создается в формате .txt или .html.

Диаграмма классов



Текст программы

using System;

using System.IO;

using System.Diagnostics;

using System.Collections.Generic;

//using EditDistanceProject;

using System.Windows.Forms;

using Liven;

using System.Threading.Tasks;

using System.Text;

namespace лаба4

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent(); //просто не будем это трогать.

}

/// <summary>

/// Список слов

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

List<string> list = new List<string>();//создали пустой лист

private void Form1\_Load(object sender, EventArgs e)

{

}

private void button1\_Click(object sender, EventArgs e)

{

OpenFileDialog fileD = new OpenFileDialog();

fileD.Filter = "текстовые файлы|\*.txt"; //ограничиваемся текстовыми файлами

if (fileD.ShowDialog() == DialogResult.OK)

{

Stopwatch t = new Stopwatch(); //штучка для измерения времени

t.Start(); //начали отсчет

string text = File.ReadAllText(fileD.FileName); //чтение файла в виде строки

string[] textArray = text.Split(new char[] { ' ', '.', ',', '!', '?', '-', ';', ':', '/', '\t', '\n' }, StringSplitOptions.RemoveEmptyEntries); //разделяем строку на слова

foreach (string i in textArray)

{

string str = i.Trim(); //поудаляли пробелы перед и после слова

if (!list.Contains(str)) list.Add(str); //если в списке такого слова нет, добавляем его

}

t.Stop(); //остановили таймер

this.label4.Text = (t.ElapsedTicks \* 0.0001).ToString(); //посчитали время

this.label1.Text = list.Count.ToString(); //посчитали количество записанных слов

}

else

{

MessageBox.Show("Нужно выбрать файл");

}

}

private void buttonSeach\_Click(object sender, EventArgs e)

{

string word = this.textBoxFind.Text.Trim(); //введенное слово для поиска

if (!string.IsNullOrWhiteSpace(word) && list.Count > 0) //если слово не пустое

{

int kol;

if (!int.TryParse(this.TextBoxKol.Text.Trim(), out kol))

{

MessageBox.Show("Необходимо указать максимальное расстояние");

return;

}

if (kol < 1 || kol > 5)

{

MessageBox.Show("Масксимальное расстояние должно быть больше 1 и меньше 5");

return;

}

int ThreadKol;

if (!int.TryParse(this.textBoxPotok.Text.Trim(), out ThreadKol))

{

MessageBox.Show("Необходимо указать количество потоков");

return;

}

Stopwatch t = new Stopwatch();

t.Start(); //начали отсчет

//----------параллельный поиск-----------

List<ParallelSearchResult> Result = new List<ParallelSearchResult>();

List<MinMax> arrayDilList = SubArrays.DivideSubArrays(0, list.Count, ThreadKol);//создали массив Минмаксов, первый элемент 0, последний = числу эл

int count = arrayDilList.Count;//количество потоков=количество элементов в массиве

Task<List<ParallelSearchResult>>[] tasks = new Task<List<ParallelSearchResult>>[count]; //массив классов с результатами (данными о каждом подмассиве)

for (int i = 0; i < count; i++) //запускаем потоки

{

List<string> tempTaskList = list.GetRange(arrayDilList[i].Min, arrayDilList[i].Max - arrayDilList[i].Min);//копируем значения из нужного диапазона

tasks[i] = new Task<List<ParallelSearchResult>>(

ThreadTaskk.ArrayThreadTask,

new ParallelSearchThreadParam()

{

tempList = tempTaskList,

maxDist = kol,

ThreadNumber = i,

wordPattern = word

});//разобрались с параметрами потока

tasks[i].Start();//запускаем поток

}

Task.WaitAll(tasks); //завершаем работу, только когда закончилась работа со всеми потоками

t.Stop();

for (int i = 0; i < count; i++)//объединение результатов

{

Result.AddRange(tasks[i].Result);

}

//-----------------------

this.labelTime2.Text = (t.ElapsedTicks \* 0.0001).ToString(); //время поиска

this.textBoxPotok.Text = count.ToString();

this.listBoxResults.Items.Clear(); //очистили список (чтоб не показывал предыдущие результаты)

this.listBoxResults.BeginUpdate();

foreach (var i in Result)

{

string ivan = i.word + " (расстояние = " + i.distance.ToString() + " поток = " + (i.ThreadNumber + 1).ToString() + ")";

this.listBoxResults.Items.Add(ivan);

}

this.listBoxResults.EndUpdate();

}

else

{

MessageBox.Show("Нужно ввести слово для поиска");

}

}

private void label5\_Click(object sender, EventArgs e)

{

}

private void buttonReport\_Click(object sender, EventArgs e)//вывод отчета

{

string TempReportFileName = "Report\_" + DateTime.Now.ToString("dd\_MM\_yyyy\_hhmmss"); //имя

SaveFileDialog fd = new SaveFileDialog();

fd.FileName = TempReportFileName;

fd.DefaultExt = ".html";

fd.Filter = "HTML Reports|\*.html";

if (fd.ShowDialog() == DialogResult.OK)

{

string ReportFileName = fd.FileName;

StringBuilder b = new StringBuilder(); //ТЁМНАЯ МАГИЯ ЗАПИСИ ТАБЛИЦЫ НЕ ДАЙ БОГ КОМУ РАЗБИРАТЬСЯ

b.AppendLine("<html>");

b.AppendLine("<head>");

b.AppendLine("<meta http-equiv='Content-Type' content='text/html; charset = UTF - 8'/>");

b.AppendLine("<title>" + "Отчёт: " + ReportFileName + "</title>");

b.AppendLine("</head>");

b.AppendLine("<body>");

b.AppendLine("<h1>" + "Отчёт: " + ReportFileName + "</h1>");

b.AppendLine("<table border='1'>");

b.AppendLine("<tr>");

b.AppendLine("<td>Время чтения из файла</td>");

b.AppendLine("<td>" + this.label4.Text + "</td>");

b.AppendLine("</tr>");

b.AppendLine("<tr>");

b.AppendLine("<td>Количество уникальных слов в файле</td>");

b.AppendLine("<td>" + this.label1.Text + "</td>");

b.AppendLine("</tr>");

b.AppendLine("<tr>");

b.AppendLine("<td>Слово для поиска</td>");

b.AppendLine("<td>" + this.textBoxFind.Text + "</td>");

b.AppendLine("</tr>");

b.AppendLine("<tr>");

b.AppendLine("<td>Максимальное расстояние для нечеткого поиск а</td>");

b.AppendLine("<td>" + this.TextBoxKol.Text + "</td>");

b.AppendLine("</tr>");

b.AppendLine("<tr>");

b.AppendLine("<td>Время нечеткого поиска</td>");

b.AppendLine("<td>" + this.label4.Text + "</td>");

b.AppendLine("</tr>");

b.AppendLine("<tr valign='top'>");

b.AppendLine("<td>Результаты поиска</td>");

b.AppendLine("<td>");

b.AppendLine("<ul>");

foreach (var x in this.listBoxResults.Items)

{

b.AppendLine("<li>" + x.ToString() + "</li>");

}

b.AppendLine("</ul>");

b.AppendLine("</td>");

b.AppendLine("</tr>");

b.AppendLine("</table>");

b.AppendLine("</body>");

b.AppendLine("</html>");

File.AppendAllText(ReportFileName, b.ToString());

MessageBox.Show("Отчет сформирован. Файл: " + ReportFileName);

}

}

}

}

Сlass 1

public class EditDistance //посчитать расстояние

{

public static int Distance(string str1, string str2)

{

if ((str1==null)||(str2==null)) return -1; //если обе строки пустые, конец

int len1 = str1.Length;//длины слов

int len2 = str2.Length;

if ((len1 == 0) && (len2 == 0)) return 0; //если есть нулевые строки, все просто

if (len1 == 0) return len2;

if (len2 == 0) return len1;

string strUp1 = str1.ToUpper();

string strUp2 = str2.ToUpper(); //приводим к верхнему регистру

int[,] matrix = new int[len1 + 1, len2 + 1];//создали матрицу

for (int i = 0; i < len1; i++) matrix[i, 0] = i;

for (int j = 0; j < len2; j++) matrix[0, j] = j; //пронумеровали столбики и строчки

for (int i=1; i<len1+1; i++)

{

for (int j=1; j<len2+1; j++)

{

int sEqual = ((strUp1.Substring(i - 1, 1) == strUp2.Substring(j - 1, 1)) ? 0 : 1);

//если соответствующие символы равны, присваиваем 0, иначе 1

int oI = matrix[i, j - 1] + 1; //insert

int oD = matrix[i - 1, j] + 1; //delete

int oR = matrix[i - 1, j - 1] + sEqual; //replace

matrix[i, j] = Math.Min(Math.Min(oI, oD), oR); //минимум

if ((i>1)&&(j>1)&& //если элемент не первый (не номер столбика)

(strUp1.Substring(i-1,1) == strUp2.Substring(j-2, 1))&& //если при перестановке символы равны

(strUp1.Substring(i - 2, 1) == strUp2.Substring(j - 1, 1)))

{

matrix[i, j] = Math.Min(matrix[i, j], matrix[i - 2, j - 2] + sEqual);

}

}

}

return matrix[len1, len2]; //вернули нижний правый

}

}

public class MinMax //разделение на подмассивы

{

public int Min { get; set; }

public int Max { get; set; }

public MinMax (int m1, int m2)

{

this.Min = m1;

this.Max = m2;

}

}

public static class SubArrays //деление массива на последовательности

{

public static List<MinMax> DivideSubArrays (int Index1, int Index2, int subArrayCount)

{

List<MinMax> result = new List<MinMax>(); //список пар и индексами подмассивов

if ((Index2-Index1)<=subArrayCount) //если число элементов недостаточно для деления,

{

result.Add(new MinMax(0, (Index2 - Index1)));//возвращаем целиком

}

else

{

int delta = (Index2 - Index1) / subArrayCount;//нашли, сколько должно быть в подмассиве элементов

int currentBegin = Index1;

while ((Index2-currentBegin)>=2\*delta)//пока размер массива укладывается в оставшуюся последовательность

{

result.Add(new MinMax(currentBegin, currentBegin + delta));//делаем подмассив

currentBegin += delta;//сдвигаем точку отсчета

}

result.Add(new MinMax(currentBegin, Index2));//оставшаяся часть массива

}

return result;

}

}

public class ParallelSearchResult //хранение результатов

{

public string word { get; set; } //найденное слово

public int distance { get; set; } //расстояние

public int ThreadNumber { get; set; } //номер потока

}

class ParallelSearchThreadParam //класс с параметрами для поиска

{

public List<string> tempList { get; set; } //массив

public string wordPattern { get; set; } //слово

public int maxDist { get; set; } //максимальное расстояние

public int ThreadNumber { get; set; } //номер потока

}

public class ThreadTaskk //поиск слов в потоке

{

public static List<ParallelSearchResult> ArrayThreadTask (object paramObj)//сам поиск

{

ParallelSearchThreadParam param = (ParallelSearchThreadParam)paramObj; //создали объект с указ параметрами

string wordUp = param.wordPattern.Trim().ToUpper(); //привели к верхнему регистру

List<ParallelSearchResult> Result = new List<ParallelSearchResult>();//новый массив для хранения результатов

foreach (string str in param.tempList) //проходимся по словам в подмассиве

{

int dist = EditDistance.Distance(str.ToUpper(), wordUp); //посчитали расстояние Ливенштейна

if (dist <= param.maxDist)//если норм,

{

ParallelSearchResult temp = new ParallelSearchResult()//записываем результаты

{

word = str,

distance = dist,

ThreadNumber = param.ThreadNumber

};

Result.Add(temp);

}

}

return Result; // и выводим результат

}

}

Экранные формы с примером выполнения программы



